PI 1159476

THE UNITED STATES OF AMERICA

TO ALL TO WHOM THESE PRESENTS SHALL COME:

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office

April 28, 2004

THIS IS TO CERTIFY THAT ANNEXED HERETO IS A TRUE COPY FROM THE RECORDS OF THE UNITED STATES PATENT AND TRADEMARK OFFICE OF THOSE PAPERS OF THE BELOW IDENTIFIED PATENT APPLICATION THAT MET THE REQUIREMENTS TO BE GRANTED A FILING DATE.

APPLICATION NUMBER: *60/493,132*
FILING DATE: *August 06, 2003*
RELATED PCT APPLICATION NUMBER: *PCT/US04/03609*

By Authority of the
COMMISSIONER OF PATENTS AND TRADEMARKS

T. WALLACE
Certifying Officer

PRIORITY DOCUMENT
SUBMITTED OR TRANSMITTED IN COMPLIANCE WITH RULE 17.1(a) OR (b)

PATENT APPLICATION SERIAL NO. _____

U.S. DEPARTMENT OF COMMERCE
PATENT AND TRADEMARK OFFICE
FEE RECORD SHEET

03/[illegible]/2003 CCHAU1 00000014 60493132
01 FC:2005 80.00 OP

PTO-1556
(5/87)

*U.S. Government Printing Office: 2002 — 489-267/59033

PTO/SB/16 (10-01)
Approved for use through10/31/2002. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE
Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

PROVISIONAL APPLICATION FOR PATENT COVER SHEET

This is a request for filing a PROVISIONAL APPLICATION FOR PATENT under 37 CFR 1.53(c).

Express Mail Label No.	EL919127921US

INVENTOR(S)

Given Name (first and middle [if any])	Family Name or Surname	Residence (City and either State or Foreign Country)
Aravind R.	Dasu	Tempe, Arizona
Ali	Akoglu	Tempe, Arizona
Sethuraman	Panchanathan	Gilbert Arizona

☐ *Additional Inventors are being named on the _____ separately numbered sheets attached hereto*

TITLE OF THE INVENTION (500 characters max)

Heterogeneous Hierarchical Routing Architecture

CORRESPONDENCE ADDRESS

Direct all correspondence to:

☑ Customer Number 28,529 → *Place Customer Number Bar Code Label here*

OR

Type Customer Number here

☐ Firm or Individual Name					
Address					
Address					
City		State		ZIP	
Country		Telephone		Fax	

ENCLOSED APPLICATION PARTS *(check all that apply)*

☑ Specification *Number of Pages* 36
☐ Drawing(s) *Number of Sheets*
☐ Application Data Sheet. See 37 CFR 1.76
☐ CD(s), Number
☑ Other (specify) return receipt postcard

METHOD OF PAYMENT OF FILING FEES FOR THIS PROVISIONAL APPLICATION FOR PATENT

☑ Applicant claims small entity status. See 37 CFR 1.27.
☑ A check or money order is enclosed to cover the filing fees
☑ The Commissioner is hereby authorized to charge filing fees or credit any overpayment to Deposit Account Number: 070135
☐ Payment by credit card. Form PTO-2038 is attached.

FILING FEE AMOUNT ($): $80.00

The invention was made by an agency of the United States Government or under a contract with an agency of the United States Government.

☑ No.
☐ Yes, the name of the U.S. Government agency and the Government contract number are: _____

Respectfully submitted,

SIGNATURE

Date 08/06/2003

TYPED or PRINTED NAME Thomas D. MacBlain

REGISTRATION NO. *(if appropriate)* 24,583

Docket Number: 9138-0106PRV3

TELEPHONE (602) 530-8088

USE ONLY FOR FILING A PROVISIONAL APPLICATION FOR PATENT

This collection of information is required by 37 CFR 1.51. The information is used by the public to file (and by the PTO to process) a provisional application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 8 hours to complete, including gathering, preparing, and submitting the complete provisional application to the PTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, D.C. 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Box Provisional Application, Assistant Commissioner for Patents, Washington, D.C. 20231.

PTO/SB/17 (01-03)
Approved for use through 04/30/2003. OMB 0651-0032
U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

FEE TRANSMITTAL for FY 2003

Effective 01/01/2003. Patent fees are subject to annual revision.

[X] Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT ($) 80.00

Complete if Known	
Application Number	
Filing Date	herewith
First Named Inventor	Dasu
Examiner Name	
Art Unit	
Attorney Docket No.	9138-0106PRV3

METHOD OF PAYMENT *(check all that apply)*

[X] Check [ ] Credit card [ ] Money Order [ ] Other [ ] None

[X] Deposit Account:

Deposit Account Number: 070135

Deposit Account Name: Gallagher & Kennedy, P.A.

The Commissioner is authorized to: *(check all that apply)*

[ ] Charge fee(s) indicated below [X] Credit any overpayments

[X] Charge any additional fee(s) during the pendency of this application

[ ] Charge fee(s) indicated below, except for the filing fee to the above-identified deposit account.

FEE CALCULATION

1. BASIC FILING FEE

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1001	750	2001	375	Utility filing fee	
1002	330	2002	165	Design filing fee	
1003	520	2003	260	Plant filing fee	
1004	750	2004	375	Reissue filing fee	
1005	160	2005	80	Provisional filing fee	80

SUBTOTAL (1) ($) 80

2. EXTRA CLAIM FEES FOR UTILITY AND REISSUE

		Extra Claims		Fee from below	Fee Paid
Total Claims		-20** =		X	=
Independent Claims		- 3** =		X	=
Multiple Dependent					=

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description
1202	18	2202	9	Claims in excess of 20
1201	84	2201	42	Independent claims in excess of 3
1203	280	2203	140	Multiple dependent claim, if not paid
1204	84	2204	42	** Reissue independent claims over original patent
1205	18	2205	9	** Reissue claims in excess of 20 and over original patent

SUBTOTAL (2) ($)

**or number previously paid, if greater; For Reissues, see above

FEE CALCULATION (continued)

3. ADDITIONAL FEES

Large Entity Fee Code	Large Entity Fee ($)	Small Entity Fee Code	Small Entity Fee ($)	Fee Description	Fee Paid
1051	130	2051	65	Surcharge - late filing fee or oath	
1052	50	2052	25	Surcharge - late provisional filing fee or cover sheet	
1053	130	1053	130	Non-English specification	
1812	2,520	1812	2,520	For filing a request for *ex parte* reexamination	
1804	920*	1804	920*	Requesting publication of SIR prior to Examiner action	
1805	1,840*	1805	1,840*	Requesting publication of SIR after Examiner action	
1251	110	2251	55	Extension for reply within first month	
1252	410	2252	205	Extension for reply within second month	
1253	930	2253	465	Extension for reply within third month	
1254	1,450	2254	725	Extension for reply within fourth month	
1255	1,970	2255	985	Extension for reply within fifth month	
1401	320	2401	160	Notice of Appeal	
1402	320	2402	160	Filing a brief in support of an appeal	
1403	280	2403	140	Request for oral hearing	
1451	1,510	1451	1,510	Petition to institute a public use proceeding	
1452	110	2452	55	Petition to revive - unavoidable	
1453	1,300	2453	650	Petition to revive - unintentional	
1501	1,300	2501	650	Utility issue fee (or reissue)	
1502	470	2502	235	Design issue fee	
1503	630	2503	315	Plant issue fee	
1460	130	1460	130	Petitions to the Commissioner	
1807	50	1807	50	Processing fee under 37 CFR 1.17(q)	
1806	180	1806	180	Submission of Information Disclosure Stmt	
8021	40	8021	40	Recording each patent assignment per property (times number of properties)	
1809	750	2809	375	Filing a submission after final rejection (37 CFR 1.129(a))	
1810	750	2810	375	For each additional invention to be examined (37 CFR 1.129(b))	
1801	750	2801	375	Request for Continued Examination (RCE)	
1802	900	1802	900	Request for expedited examination of a design application	

Other fee (specify) _____

*Reduced by Basic Filing Fee Paid SUBTOTAL (3) ($)

SUBMITTED BY			(Complete *(if applicable)*)
Name (Print/Type)	Thomas D. MacBlain	Registration No. (Attorney/Agent) 24,583	Telephone 602-530-8088
Signature			Date 8/6/2003

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

This collection of information is required by 37 CFR 1.17 and 1.27. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, Washington, DC 20231. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, Washington, DC 20231.

If you need assistance in completing the form, call 1-800-PTO-9199 (1-800-786-9199) and select option 2.

Attorney Docket No. 9138-0106PRV3

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Dasu et al.

Filed: Herewith

Title: **Heterogeneous Hierarchical Routing Architecture**

CERTIFICATE OF MAILING BY EXPRESS MAIL
"Express Mail" mailing label number EL919127921US

Mail Stop Provisional Patent Application
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Dear Commissioner:

I hereby certify that the following correspondence is being deposited in the United States Postal Service as Express Mail on the date shown below in an envelope addressed as shown above.

1. Provisional Application for Patent Cover Sheet (1 sheet);
2. Fee Transmittal for FY 2003 (1 sheet in duplicate);
3. Specification (36 pages plus cover sheet);
4. Check for $80.00; and
5. A return receipt postcard.

8/6/03
Date

Suzanne Shields

GALLAGHER & KENNEDY, P.A.
2575 East Camelback Road
Phoenix, Arizona 85016-9255
Tel. No. (602) 530-8000
Fax. No. (602) 530-8500

Atto[redacted] Docket No. 9138-0106PRV3

Express Mail Label No. EL919127921US

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Provisional Patent Application

Title: Heterogeneous Hierarchical Routing Architecture

Inventors: Aravind R. Dasu, Tempe, Arizona
Ali Akoglu, Tempe, Arizona
Sethuraman Panchanathan, Gilbert, Arizona

Attorneys for Applicant: Thomas D. MacBlain
Gallagher & Kennedy
2575 East Camelback Road
Phoenix, AZ 85016-9225

Heterogeneous Hierarchical Routing Architecture

1 Physical Design Cycle

This chapter talks about the Physical Design Cycle for VLSI and FPGAs. The VLSI chip design cycle includes the steps of system specification, functional design, logic design, circuit design, physical design, fabrication and packaging. The physical design automation of FPGA involves three steps, which include partitioning, placement, and routing. The VLSI and FPGA design cycles follow brief introduction.

1.1 Introduction

Despite advances in VLSI design automation, the time it takes to market for a chip is unacceptable for many applications. The key problem is time taken due to fabrication of chips and therefore there is a need to find new technologies, which minimize the fabrication time. Gate Arrays use less time in fabrication as compared to full custom chips since only routing layers are fabricated on top of pre-fabricated wafer. However fabrication time for gate arrays is still unacceptable for several applications. In order to reduce the time to fabricate interconnects; programmable devices have been introduced which allow users to program the devices as well as interconnect.

FPGA is a new approach to ASIC design that can dramatically reduce manufacturing turn around time and cost. In its simplest form an FPGA consists of regular array of programmable logic blocks interconnected by a programmable routing network. A programmable logic block is a RAM and can be programmed by the user to act as a small logic module. The key advantage of FPGA is re-programmability.

In this section we first talk about the classical VLSI Physical Design cycle and then introduce the physical design cycle for FPGAs.

The VLSI chip design cycle includes the steps of system specification, functional design, logic design, circuit design, physical design, fabrication and packaging. We focus on the physical design cycle. First we give definitions of the steps of partitioning, floor planning, placement, routing and compaction that are included in physical design then we explain the routing step in more detail with different approaches of routing algorithms and their comparisons.

The physical design automation of FPGA involves three steps, which include partitioning, placement, and routing. Partitioning problem in FPGAs is significantly different than the partitioning problem s in other design styles. This problem depends on the architecture in which the circuit has to be implemented. Placement problem in FPGAs is very similar to the gate array placement problem. The routing problem in FPGAs is to find a connection path and program the appropriate interconnection points. We will discuss the architecture of FPGAs and their physical design cycle.

1.2 Physical Design Cycle for VLSI

In this step the circuit representation of each component is converted into a geometric representation. This representation is a set of geometric patterns, which perform the intended logic function of the corresponding component. Connections between different components are also expressed as geometric patterns. Physical design is a very complex process and therefore it is usually broken into various subsets.

The input to the physical design cycle is the circuit diagram and the output is the layout of the circuit. This is accomplished in several stages such as partitioning, floor planning, placement, routing and compaction. Each of these stages will be discussed in detail but to give an overview, a brief description of all stages are given here.

1.2.1 Partitioning

A chip may contain several transistors. Layout of the entire circuit cannot be handled due to the limitation of memory space as well as computation power available. Therefore it is normally partitioned by grouping the components into blocks. The actual partitioning process considers many factors such as the size of the blocks, number of blocks, and the number of interconnections between the blocks. The set of interconnections required is referred as a net list. In large circuits the partitioning process is hierarchical and at the topmost level a chip may have 5 to 25 blocks. Each block is then partitioned recursively into smaller blocks.

1.2.2 Floor planning and Placement

This step is concerned with selecting good layout alternatives for each block as well as the entire chip. The area of each block can be estimated after partitioning and is based approximately on the number and type of commonness in that block. In addition interconnect area required within the block must also be considered. Very often the task of floor plan layout is done by a design engineer rather than a CAD tool due to the fact that human is better at visualizing the entire floor plan and take into account the information flow. In addition certain components are often required to be located at specific positions on the chip. During placement the blocks are exactly positioned on the chip. The goal of placement is to find minimum area arrangement for the blocks that allows completion of interconnections between the blocks while meeting the performance constraints. Placement is usually done in two phases. In the first phase initial placement is done. In the second phase the initial placement is evaluated and iterative improvements are made until layout has minimum area or best performance.

The quality of placement will not be clear until the routing phase has been completed. Placement may lead to un-routable design. In that case another iteration of placement is necessary. To limit the number of iterations of the placement algorithm an estimate of the required routing space is used during the placement process. A good routing and circuit performance heavily depend on a good placement algorithm. This is due to the fact that once the position of the block is fixed; there is not much to do to improve the routing and the circuit performance.

1.2.3 Routing

The objective of routing is to complete the interconnection between the blocks according to the specified net list. First the space that is not occupied by the blocks (routing space) is partitioned into rectangular regions called channels and switchboxes. This includes the space between the blocks. The goal of the router is to complete all circuit connections using the shortest possible wire length and using only the channel and switch boxes. This is usually done in two phases referred as global routing and detailed routing phases. In global routing connections are completed between the proper blocks disregarding the exact geometric details of each wire. For each wire global router finds a list of channels and switchboxes to be used as passageway for that wire. Detailed routing that completes point-to-point connections follows global routing. Global routing is converted into exact routing by specifying the geometric information such as location and spacing of wires. Routing is a very well defined studied problem. Since almost all routing problems are computationally hard the researchers have focused on heuristic algorithms.

1.2.4 Compaction

Compaction is the task of compressing the layout in all directions such that the total area is reduced. By making the chip smaller wire lengths are reduced which in turn reduces the signal delay.

1.3 Global Routing

This section will discuss the approaches to global routing problem. Generally these approaches are classified as sequential and concurrent approaches.

1.3.1 Sequential approach

In this approach nets are routed one by one. If a net is routed it may block other nets which are to be routed. As a result this approach is very sensitive to the order of the nets that are considered for routing. Usually the nets are ordered with respect to their criticality. The criticality of a net is determined by the importance of the net. For example a clock net may determine the performance of the circuit so it is considered highly critical. However sequencing techniques don't solve the net ordering problem satisfactorily. An improvement phase is used to remove blockages when further routing is not feasible. This may also not solve the net ordering problem so in addition to that 'rip-up and reroute' technique [Bol79, DK82] and 'shove-aside' techniques are used. In rip-up and reroute the interfering wires are ripped up and rerouted to allow routing of affected nets. Whereas in shove aside technique wires that allow completion of failed connections are moved aside without breaking the existing connection. Another approach [De86] is to first route simple nets consisting of only two or three terminals since there are few choices for routing such nets. After the simple nets are routed, a Steiner Tree algorithm is used to route intermediate nets. Finally a maze routing algorithm is used to route the remaining multi-terminal nets that are not too numerous.

The sequential approach includes:

a) Two terminal algorithms:
    i. Maze routing algorithms
    ii. Line probe algorithms
    iii. Shortest path based algorithms
b) Multi terminal algorithms:
    i. Steiner tree based algorithms.

1.3.2 Concurrent approach

This approach avoids the ordering problem by considering the routing of all nets simultaneously. This approach is computationally hard and no efficient polynomial algorithm is known even for two terminal nets. As a result integer method programming methods have been suggested. The corresponding integer program is usually too large to be employed efficiently. As a result hierarchical methods that work top town are used to partition the problem into smaller sub problems, which can be solved by integer programming.

1.3.3 Maze Routing Algorithms

[Lee61] introduced an algorithm for routing a two terminal net on a grid in 1961. This algorithm and its variations form the class of maze routing algorithms. Maze routing algorithms are used to find a path between pair of points called the source(s) and target (t) in a planar rectangular grid graph. The areas available for routing are represented as unblocked vertices. The objective of a maze algorithm is to find a path between the source and target vertex without using any blocked vertex. The process begins with the exploration phase in which several paths start at the source and are expanded until one of them reaches the target. Once the target is reached the vertices need to be retraced to the source to identify the path.

1.3.3.1 Lee's Algorithm:

The key popularity of Lee's maze algorithm is its simplicity and its guarantee of finding an optimal solution if one exists. The exploration phase is an improved version of breadth first search. Search is like a wave propagating from the source. The source is labeled 0 and the wave propagates to all unblocked vertices adjacent to the source. Every unblocked vertex adjacent to the source is marked by 1. Then every unblocked vertex adjacent to the vertices with a label 1 is marked with label 2 and so on. This process continues until the target is reached or no further expansion is possible. The path found is guaranteed to be the shortest path as well. This algorithm requires a large number of storage space and its performance degrades rapidly when the size of grid increases.

1.3.3.2 Soukup's algorithm:

Lee explores the grid symmetrically searching equally in the directions away from the target. It requires a large search time. Soukup [Sou78] proposed an iterative algorithm to overcome this. During each iteration algorithm explores the direction toward the target without changing the direction until it reaches the target otherwise it goes away from the target. If the target is reached the exploration phase ends. If target is not reached the

search is conducted iteratively. If the search goes away from the target, algorithm changes the direction and a new iteration begins. This algorithm improves the speed of Lee's algorithm by a factor of 10-50. It guarantees finding a path if exists, however it may not be the shortest path..

1.3.3.3 Hadlock's Algorithm:

An alternative approach to improve the speed was proposed by Hadlock [Had75]. Algorithm is called minimum detour algorithm. According to Hadlock, the length of a path (P) connec5tting a source and target can be given by (M(s, t) + 2d(P), where M(s, t) is Manhattan distance between source and target and d (P) is the number of vertices on path P that are directed away from the target. Length of P is minimized if d is minimized since M(s, t) is constant for any sour target pair. The exploration phase uses the detour number instead of labeling the wave front number. The detour number of a path is the number of times that the path has tuned away from the target.

1.3.3.4 Comparison of Maze Routing Algorithms

These algorithms are grid-based methods. The time and space required by these algorithms depend linearly on their search space. Lee's algorithm guarantees finding a shortest path between any vertices if a path exist, however worst case occurs when source is located at the center and the target is located at a corner of routing area. In this case all vertices have to be scanned before target is reached. Soukup's algorithm solves the shortcoming of breadth first search by using depth-first search until an obstacle is seen. When obstacle is reached then breadth first methods is used to get around it. Search time is smaller with the nature of depth-first search, however shortest path is not guaranteed. Worst case occurs when search goes in the direction of target, which is opposite direction of the passageway through the obstacle. Hadlock's algorithm is a breadth-first search method. The difference from Lee's algorithm is the way wave front is labeled. In Hadlock's algorithm search can prefer the direction toward the target to direction away from the target. Search time is shorter than Lee's algorithm. Worst case occurs when search goes toward the target and opposite the passageway through the obstacle.

All maze routers are grid-based methods. Information must be kept for each grid node so a large memory space is needed for large grid. As an approximate estimate for a chip size of 10000λ x 10000λ requires as much as 350Mbytes of memory and 66 seconds to route on net on a 15MIPS workstation. There may be 5000 to 10000 nets in a typical chip. In order to reduce the large memory requirements and run times line-probe algorithms were developed.

1.3.4 Line-Probe Algorithms

These algorithms were developed by Mikami, Tabuchi [MT68] and Hightower [Hig69]. Line-probe algorithm reduces the size of memory requirement by using line segments instead of grid nodes in the search. The time and space requirements are in the order of O(L), where L is the number of line segments produced by the algorithm.

Initially lists 'slist' and 'tlist' contain the line segments generated from the source and target respectively. The generated line segments don't pass through any obstacle. During

each iteration new line segments are generated and appended to the corresponding list. If a line segment from *slist* intersects with a line segment from *tlist* then the exploration ends otherwise exploration continues iteratively. Retracing the line segments in tlist, starting from the target going through the intersection and then retracing the line segments in the slist until source is reached can form the path. In Mikami's algorithm every grid node on the line segment is an escape point to generate new perpendicular segments. Hightower's algorithm uses only one escape point on a line segment. The disadvantage is that it may not be able to find a path joining two points.

1.3.5 Steiner Tree Based Algorithms

Routing algorithms explained so far are not suitable for routing the multi-terminal nets. Different techniques have been proposed that modify maze routing and line-probe algorithms. In one approach multi terminal nets are decomposed into several two terminal nets and those two terminal nets are routed. The quality of the routing in this approach depends on the decomposition of the nets that might result in sub-optimal routes. The natural approach for multi terminal routing is Rectilinear Steiner Trees (RST) approach. RST is a tree with rectilinear edges. Length of the tree is the sum of the lengths of all edges in the tree.

1.4 Physical Design Automation of FPGAs

Despite advances in VLSI design automation, the time it takes to market for a chip is unacceptable for many applications. The key problem is time taken due to fabrication of chips and therefore there is a need to find new technologies, which minimize the fabrication time. Gate Arrays use less time in fabrication as compared to full custom chips since only routing layers are fabricated on top of pre-fabricated wafer. However fabrication time for gate arrays is still unacceptable for several applications. In order to reduce the time to fabricate interconnects, programmable devices have been introduced which allow users to program the devices as well as the interconnect.

FPGA is a new approach to ASIC design that can dramatically reduce manufacturing turn around time and cost. In its simplest form an FPGA consists of regular array of programmable logic blocks interconnected by a programmable routing network. A programmable logic block is a RAM and can be programmed by the user to act as a small logic module. The key advantage of FPGA is re-programmability.

The physical design automation of FPGA involves three steps, which include partitioning, placement, and routing. Partitioning problem in FPGAs is significantly different than the partitioning problem s in other design styles. This problem depends on the architecture in which the circuit has to be implemented. Placement problem in FPGAs is very similar to the gate array placement problem. The routing problem in FPGAs is to find a connection path and program the appropriate interconnection points. We will discuss the architecture of FPGAs and their physical design cycle.

1.4.1 FPGA Technologies

FPGA architecture mainly includes two parts: the logic blocks and the routing network. A logic block has fixed number of inputs and one output. A wide range of functions can

be implemented using a logic block. Given a circuit to be implemented using FPGAs, it is first decomposed into smaller sub-circuits such that each of the sub-circuit can be implemented using a single logic block. There are two types of logic blocks. The first type is based on look-up tables while the second is based on multiplexes.

1.4.1.1 Look-up table based logic blocks:

A lookup table based logic block is just a segment of RAM. A function can be implemented by simply loading its lookup table into the logic block at power up. If function

F = \ABC+A\B\C needs to be implemented then its truth table shown below is loaded into the logic block.

A	B	C	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 1 Logic Table

In this way on receiving a creating set of inputs the logic blocks simply lookup the appropriate output and set the output line accordingly. Because of the reconfigurable nature of the lookup table based logic blocks are also called the Configurable Logic Blocks (CLB).

1.4.1.2 Multiplexer Based Logic Blocks:

Typically a multiplexer based logic block consists of three 2-to-1 multiplexers and one two input OR gate.

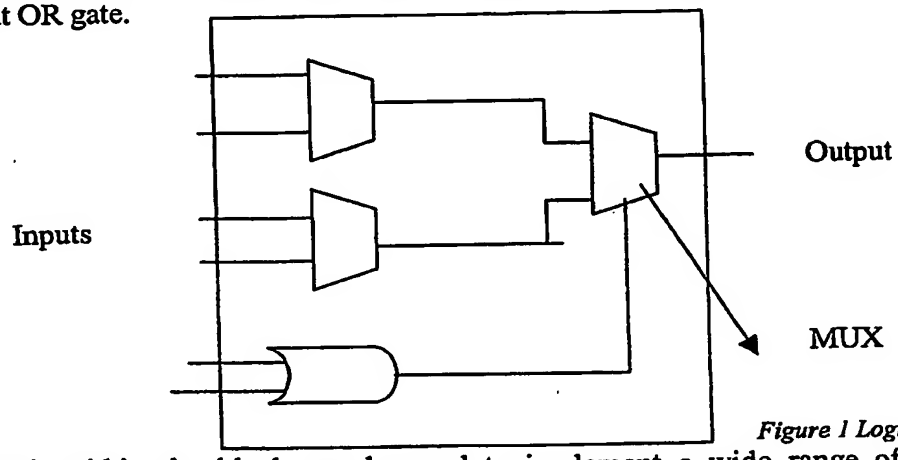


Figure 1 Logic Block

The circuit within the block can be used to implement a wide range of functions. Programming of multiplexer based logic block is achieved by routing different inputs into

the block. There are two different models of routing network: the segmented and the non-segmented.

1.4.1.2.1 Non-Segmented model

The non-segmented model is set up as regular grid of five horizontal and five vertical metal lines passing between switch blocks (S). Switch blocks are used to connect the wiring segments in one channel segment to those in another. Depending on the topology of S block each wiring segment on one side of S may be switchable to the either all or some fraction of wiring segments on each side of S block. The fewer the wiring segments a wiring segment can be switched the harder the FPGA is to be routed. In addition to the S blocks there are the connection blocks that are used to connect the logic block pins to the routing channels. Depending on the topology each L block pin may be switchable to either all or some fraction of wiring segments that pass through control block.

1.4.1.2.2 Segmented model:

The tracks in the channels contain predefined wiring segments of the same or different lengths. Other wiring segments pass though the channels vertically. Each input and output of logic block is connected to a dedicated vertical segment. As a result there are no vertical constraints. There are additional global vertical lines that provide connections between different channels. Connection between two horizontal segments is provided through an anti-fuse whereas the connection between a horizontal segment and a vertical segment is provided through a cross fuse. Programming one of these fuses provides a low resistance bi-directional connection between two segments. When blown anti-fuses connect the two segments to form a longer one.

The segmented model is uniform if the segments in all tracks have the same length and the anti-fuses in different tracks in a channel are aligned in columns. The segmented model has advantage over non-segmented model in terms of utilization of routing resources. In the non-segmented model only one segment of one net can be routed on a track. In segmented model, the segments of several nets can be assigned to track as long as no two net segments are assigned to the same track segment. The total number of programmable switches in the segmented model is higher as compared to the number of switches in the non-segmented model. The delay of a net is proportional to the number of programmable switches used. In segmented module this number is higher. As a result non-segmented model is preferred over segmented module when the performance is primary objective.

1.5 Physical Design Cycle for FPGAs

1.5.1 Partitioning

In this step the circuit to be mapped into the FPGA has to be partitioned into smaller sub circuits such that each sub circuit can be mapped to a programmable logic block. Unlike the partitioning in the other design styles, there are no constraints on the size of partition. However there are constraints on the inputs and outputs of a partition. This is the unique architecture of FPGAs.

1.5.2 Placement

In this step of the design cycle, the sub circuits, which are formed in the partitioning phase, are allocated physical locations on the FPGA, i.e., the logic block on FPGA is programmed to behave like the sub circuit that is mapped to it. This placement must be carried out in a manner that the routers can complete the interconnection. This is very critical as the routing resources of the FPGA are limited. The placement algorithms for general gate arrays are normally used for the placement in FPGAs.

1.5.3 Routing

In this phase all the sub circuits, which have been programmed on the FPGA blocks, are interconnected by blowing fuses between routing segments to achieve the interconnections.

1.6 Routing in FPGA

In this section we discuss FPGA routing for different models.

1.6.1 Routing Algorithm for the Non-Segmented Model

In this section we discuss the algorithm presented by Brown [BRV92]. The routing is completed in two steps.

1.6.1.1 Global Routing

Using a global router for standard cell designs can do global Routing in FPGA. In general such a global router divides the multi-terminal nets into two terminal nets and routes them with minimum distance path. While doing so it also tries to balance the densities by distributing the connections among the channels. The global route defines a course route for each connection by assigning it a sequence of channel segments. Figure 2.1a shows a sequence of channel segments that a global route might choose to connect some pin of logic block at grid location 4,1 to another 0,1. The global route is also called as a course grind graph. Course grid graph gives a path between two L nodes through a sequence of S and C nodes.

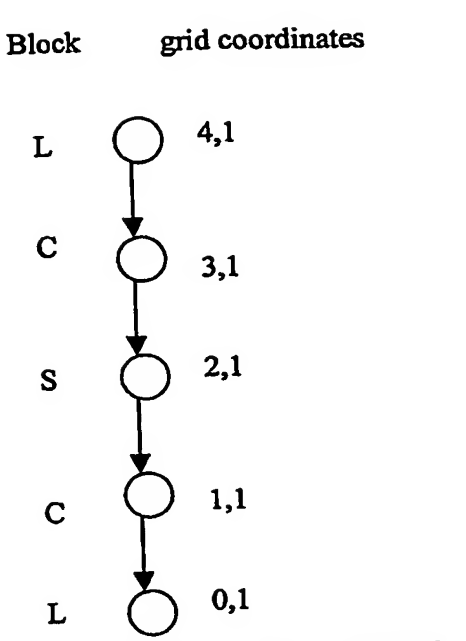


Figure2.1a Coarse grid graph

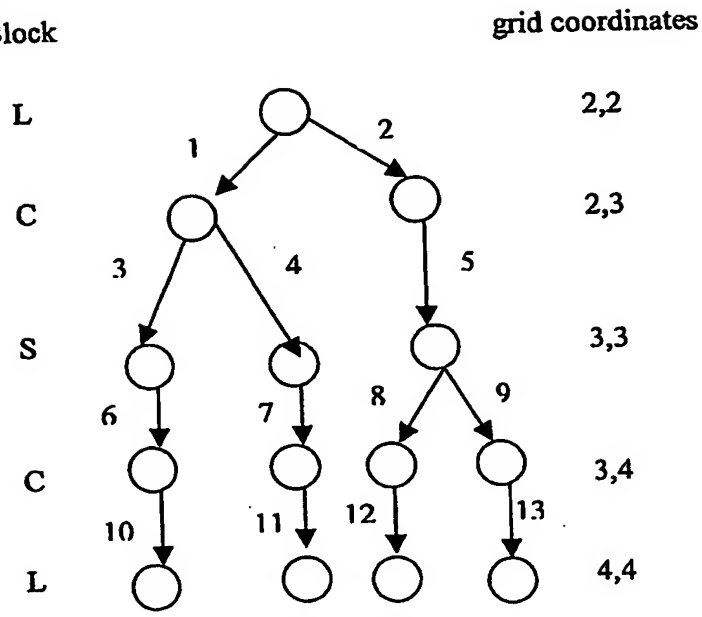


Figure 2.1b Expanded graph

1.6.1.2 Detailed Routing

Given a coarse grid graph G=(V, E) for a two terminal net, the objective of the detailed router is to choose specific wiring segments in each channel segment assigned during global routing. This is achieved in two steps:

1.6.1.2.1 Expansion of coarse grid graph

In this step a coarse grid graph is expanded to record a subset of possible ways of implementing the connection. The expansion is carried out while spanning the graph in depth first search order manner. Figure 2.1b shows the expanded graph.

```
Algorithm Graph-Expansion (G)
{
  Gd = G;
  While (DFS_COMPLETE (Gd) == FALSE)
  {
    Vi = CURRENT_DFS_VST (Gd)
    Ii = WIRE_SEGMENT (Vi, Gd)
    If (NODE_TYPE (Vi) == C || (NODE_TYPE (Vi) == S)
    {
      Vj = SUCCESS (Gd,Vi);
      Tj = SUBTREE (Gd,Vj);
    }
    if ( NODE_TYPE (Vi) == c )
    {
      for (each wiring segment l in Fc(Vi,Vj,l) )
```

```
        {
                T = DUPLICATE (Tj);
                CONNECT (Vi,Vj,T,l);
        }
          DELETE(Gd,Tj);
    }
     If( NODE_TYPE(Vi) == S )
    {
        For (each wiring segment l in Fs (Vi, Vj, l), Vk)
        {
                T = DUPLICATE (Tj);
                CONNECT (Vi, Vj, T, l);
        }
          DELETE (Gd, Tj);
    }
} /* End of while */
}
```

The description of the algorithm is as follows:

Function DFS_COMPLETE (D0 returns true if all the nodes in D are visited during depth first search. Function CURRENT_DFS_VISIT (Gd) returns the node being visited during DFS. Function WIRE_SEGMENT (Vi, Gd) returns a wire segment that connects Vi to its predecessor. Function SUBTREE (Gd, Vj) returns the sub-tree of Gd rooted by Vj. Function NODE_TYPE (Vi) returns the block type (C, S or L). If Vj is a C node Vk is the successor of Vj and a wire segment l is used to connect the two, then the function Fc (Vi, Vj, l) returns a set of wing segments that can be used to connect Vj to Vk. Similarly if Vj is an S node Vk is its successor and a wire segment l is used to connect the two then the function Fs (Vi, Vj, l, Vk) returns the set of wiring segments that can be set to connect Vj to Vk. Function DUPLICATE (T) returns a copy of tree T. CONNECT (Vi, Vj, T, l) connects the nodes Vi and Vj in T by a directed edge from Vj to T and labels connecting edge by l. DELETE (G, T) deletes the sub-tree T from G.

1.6.1.2.2 Connection Formation

The expanded graph Gd (Vd,Ed) contains a number of alternative paths. In this step all these paths are enumerated, their cost is computed and the minimum cost path is selected to implement the connection. Cost of a path is the summation of the cost of edges on that path. The cost of an edge consists of two parts: Cf (l) accounts for the competition between different nets for the same wiring segments and Cl (l) reflects the routing delay associated with the routing segment.

1.6.2 Routing Algorithms for the Segmented Model

1.6.2.1 Basic Algorithm

Green, Kaptanoglu, and Gamal have presented the algorithm for routing in segmented model [GKG93]. The input to the routing problem is a set of intervals T={I1, I2...In), a

set of tracks T = {T1, T2...Tn}. Each track Ti in T extends from column from 1 to column N, and is divided into a set of contiguous segments separated by switches. These switches are placed between two successive segments. For each interval Ii in T, LEFT (Ii) and RIGHT9Ii) are defined to be the leftmost and rightmost column in which the interval is present. The intervals in I are assumed to be sorted on their left edges. If an interval Ii is assigned to track Tj, then the segments in track Tj that are present in the columns spanned by the interval are considered occupied. A routing of I consists of an assignment of each interval Ii in I to a track such that no segment is occupied by more than one connection. The routing in the segmented model can be achieved using the algorithm SEG_ROUTER () sown below.

```
Algorithm SEG_ROUTER (I, T, A)
{
  Input:I, T, output: A

  For (i=1;i<n; i++)
      For (i=1;i<n; i++)
       s = GET_SEGMENT (j,LEFT(Ii));
       If (OCCUPIED(s) == FALSE
       {
              A [I] = j;
              MARK_OCCUPIED (Ii, Tj)
       }
}
```

SEG_ROUTER algorithm is a modified left edge algorithm. The input to the algorithm is the set of intervals I, the set of track T, whereas the output is an array A, such that A [I] gives the number of tracks T on which the interval Ii is routed. In the algorithm GET_SEGMENT (j, C) returns a segment s on track Tj such that column C is in the span of s. Function OCCUPIED(s) returns true if the segment s is occupied. MARK_OCCUPIED (Ii, Tj) marks all the segments on tracks Tj that are occupied by Ii.

1.6.2.2 Routing Algorithm for Staggered Model

The segmented model can be improved in several ways. In this model a channel is partitioned into several regions. Each region is characterized by the segment length. The tracks in each region have equal length segments separated by staggered placement of anti-fuse switches. There are three parameters with respect to the new model: number of regions (p), number of tracks (t) and the length of segment in each region (l). Determination of these tree parameters is an important step in this segmentation scheme. These parameters can be determined by a detailed empirical analysis on several standard benchmarks [BKS92]. If the length of segments in all the regions is same then the model is called as the uniform staggered model otherwise it is non-uniform staggered model.

Algorithm SEG_ROUTER can be used for routing in the staggered models. In a uniform segmented model the delay of a net is same irrespective of the routing track. Whereas, in the staggered model the delay of a net is dependent on the routing track as the number of antifuses in the path of a net in different tracks may be different. The algorithm

SEG_ROUTER is not suitable for the high performance routing as it is not sufficient to just minimize delay based on the anti-fuse elements., but also capacitance effects due to unused portion of the segments spanned by a net segment and the un programmed switches must be considered.

The algorithm starts routing the longest net first. This ensures that the delay due to the longest net is minimized which is a prerequisite for the high performance routing system. For each net it finds out a track on which the net can be routed with minimum delay. The original algorithm has three phases region selection, track selection and the region reselection. In algorithm FSCR shown below, the function OK_TO_ASSIGN (Ii, Tj) returns true if all the segments spanned by the interval Ii on track Tj are unoccupied. Function COMPUTE_DELAY (Ii, Tj) computes the delay in the interval Ii if Ii is routed based on the track Tj. Function MARK-OCCUPIED (Ii, Tj) is same as that use in algorithm SEG_ROUTER.

```
Algorithm FSCR (I, T, A)
{
      Input:I, T
      Output: A
  For (I=1;I<n;I++)
  {
      selected_track = 0;
      mimimum_delay = INFINITY;
      For (j=1;j<m;j++)
      {
         If (OK_TO_ASSIGN (Ii,Tj) == TRUE)
         {
             current_delay = COMPUTE_DELAY(Ii,Tj);
             If ( minimum_delay > current_delay )
             {
                    minimum_delay = current_delay;
                    seleted_track = j;
             }
          }
          If (selected track! = 0)
          {
             A [I]=j;
             MARK_OCCUPIED(Ii,Tj);
          }
          else
             exit /* routing not possible */
}
```

1.7 Conclusion

With emerging multimedia applications incorporating a wider flexibility of operations, there is an exponential increase in the demand for processing power. The complexity, variety of techniques and tools, and the high computation, storage and I/O bandwidths associated with multimedia processing pose several challenges, particularly from the points of scalability, resource utilization (in terms of area and energy) and real-time implementation. A number of implementation strategies have been proposed for processing multimedia data. These approaches can be broadly classified into general purpose and special purpose. Although general purpose processors with multimedia extended instruction sets pack sufficient power to handle media applications, yet their power consumptions ranging from 12 watts to 74 watts do not make them feasible for mobile computing. These processors employ mechanisms such as out of order issue and dynamic branch prediction that are costly in terms of area and power. Hence there arose a need for new high performance processors for multimedia applications. The special purpose programmable processors exploit the redundancies involved in media processing algorithms through the use of multiple floating point and media specific execution units by adding the flavors of VLIW, SIMD onto the processing core: In the programmable scenario, a data path that has already been on chip can execute only a specific set of operations. Any algorithm that does not have those specific operations needs to be interpreted in terms of those instructions. This consumes time and power. Special purpose programmable processors have higher performance with lesser control circuit complexity. They also have features to support operations on parallel-packed data, which exploits parallelism to a large extent in media processing. However the design and debugging phases involved in designing such application specific programmable processors, tends to drive the development cost higher. MPEG-4 and JPEG 2000 offer high interactivity to the user, which translates to a dynamic change in the computing resources at both the encoder and decoder units. Current technologies fall short of providing low cost and flexible solutions for multimedia processing. These drawbacks have lead into exploration of the reconfigurable architecture design space. The reconfigurable computing devices should be able to adapt the underlying hardware dynamically in response to changes in the input data or processing environment.

FPGAs are being used as a new approach to ASIC design, which offers dramatic reduction in manufacturing turnaround time and cost. The physical design cycle of a FPGA consists of three steps, partitioning, placement and routing. The FPGA partitioning problem is different from the conventional area-partitioning problem in the sense that it depends on the architecture in which the circuit has to be implemented. Placement problem is equivalent to the general gate array placement problem. However because of the segmented nature of the FPGA channels the routing considerations are quite different. In high performance FPGA designs the number of anti-fuse elements along with the unused tracks and antifuses must be given due to considerations as part of the routing phase. To match the needs of the future multimedia applications, we have proposed the first of a series of tools intended to help in the design and development of a dynamically reconfigurable multimedia processor.

2 Tool set

2.1 Profiler, Partitioner, Placement and Routing

Our research effort aids the design of a dynamically reconfigurable processor through the use of a set of analysis and design tools. These are intended to help hardware and system designers arrive at optimal hardware software co-designs for applications of a given class. The reconfigurable computing devices thus designed will be able to adapt the underlying hardware dynamically in response to changes in the input data or processing environment. The methodology for designing a reconfigurable media processor involves hardware-software co-design based on a set of three analysis and design tools[AK02]. First tool handles cluster recognition, extraction and a probabilistic model for ranking the clusters. Second tool, provides placement rules and feasible routing architecture. Third tool provides rules for data path, control units and memory design based on the clusters and their interaction. With the use of all three tools, it will be possible to design media processors that can dynamically adapt at both the hardware and software levels in embedded applications. The input to the first tool is a compiled version of the application source code. Regions of the data flow graph obtained from the source code, which are devoid of branch conditions, are identified as zones.Clusters are identified in the zones, by representing candidate instructions as data points in a multidimensional vector space. Properties of an instruction, such as location in a sequence, number of memory accesses, floating or fixed-point computation etc., constitute the various dimensions. As shown in Figure-3 clusters obtained from the previous tool are placed and routed by Tool #2, according to spatial and temporal constraints (Figure 4). The processor (of the compiler) can be any general purpose embedded computing core such as an ARM core or a MIPS processor These are RISC cores and hence are similar to general purpose machines such as UltraSPARC The output of the tool is a library of clusters and their interaction. (A Cluster comprises of sequential but not necessarily contiguous assembly level instructions). The clusters represent those groups or patterns of instructions that occur frequently and hence qualify for hardware implementation. To maximize the use of reconfigurability amongst clusters, possible parallelism and speculative execution possibilities must be exploited. The rest of this chapter explains the steps included in the partitioner, profiler tool.

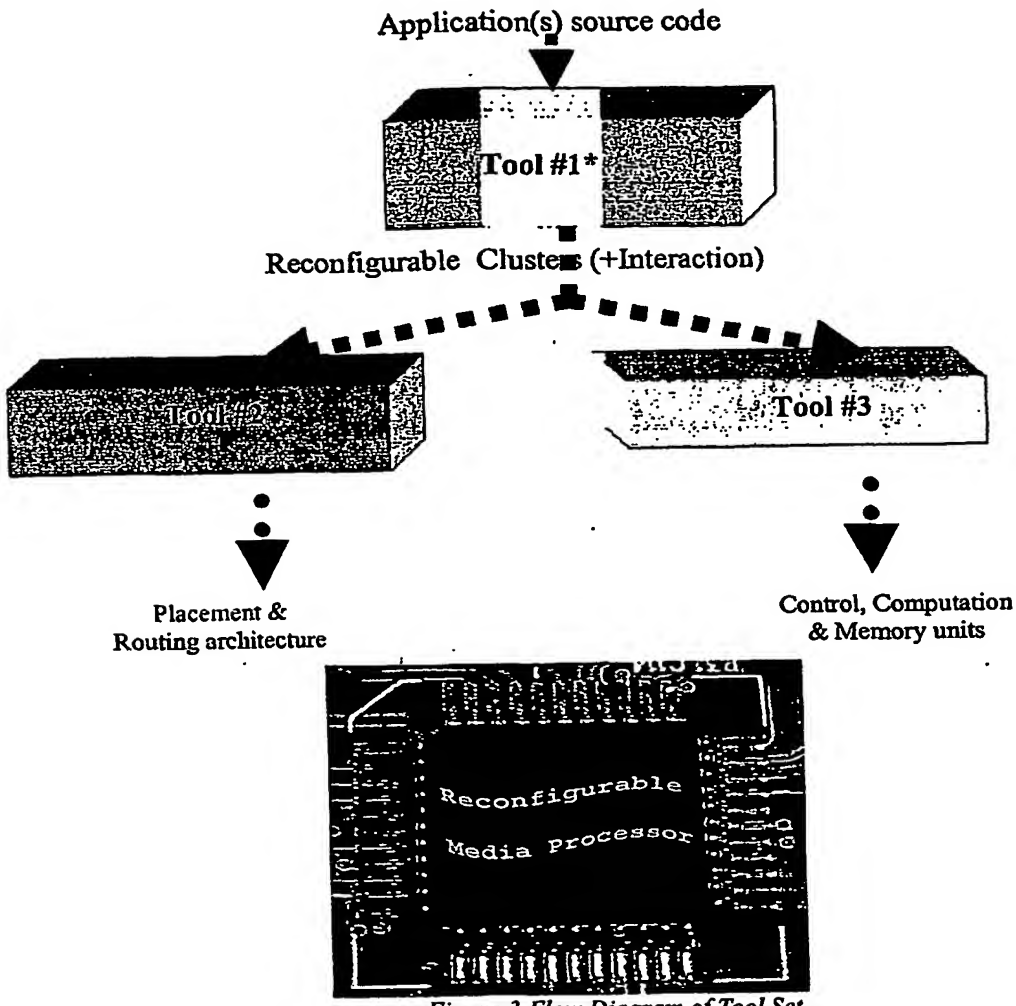


Figure 3 Flow Diagram of Tool Set

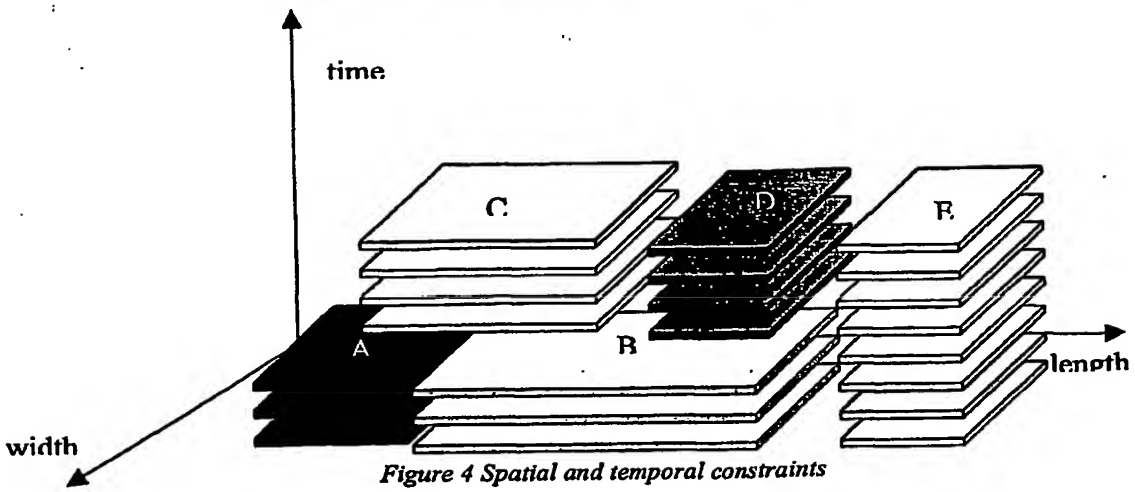


Figure 4 Spatial and temporal constraints

3 A Framework for the Design of the Routing Architecture of a Dynamically Reconfigurable Media Processor

3.1 Introduction

Commercially available FPGAs have used symmetrical, row-based, sea-of-gates or hierarchical routing architectures. We have recently proposed a tool set that will aid the design of a dynamically reconfigurable processor through the use of a set of analysis and design tools. In this paper we propose a heterogeneous hierarchical routing architecture. Compared to hierarchical and symmetrical FPGA approaches the building blocks are of variable size. Due to variable traffic among clusters and non-symmetric characteristics, different types of switches are needed at each hierarchy. Switches are variable even at the same hierarchy level. This results in heterogeneity between groups of building blocks at the same hierarchy level as opposed to classical H-FPGA approach. The aim of this paper is to show a framework for the design of the heterogeneous hierarchical routing architecture.

3.2 Routing Architectures

First we look at the existing routing architectures and then explain the proposed architecture. Routing architecture consists of wiring segments, switch boxes and the building blocks. Commercially available FPGAs have used symmetrical, row-based, sea-of-gates or hierarchical routing architectures (Figure 5). Xilinx and QuickLogic use symmetrical, Actel and Cross-Point use row based, Plessey, Algotronix use sea-of-gates, Altera and AMD use hierarchical routing architecture (Table 2).

Commercial FPGAs

Company	Architecture	Logic Block Type	Programming Technology
Actel	Row-based	Multiplexer-Based	anti-fuse
Altera	Hierarchial-PLD	PLD Block	EPROM
QuickLogic	Symmetrical Array	Multiplexer-Based	anti-fuse
Xilinx	Symmetrical Array	Look-up Table	Static RAM

Table 2

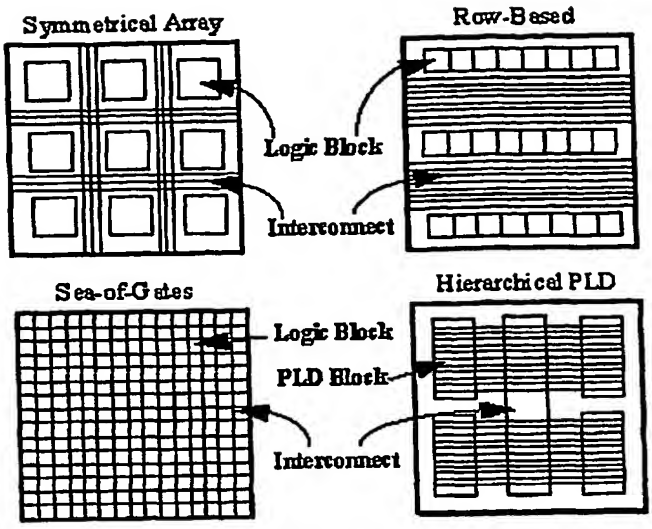


Figure 5 Routing Architecture Overview

In all of these FPGAs the interconnections and how they are programmed vary. Currently there are four technologies in use. They are: static RAM cells, anti-fuse, EPROM transistors, and EEPROM transistors. Depending upon the application, one FPGA technology may have features desirable for that application (Table 3).

Static RAM Technology -- In the Static RAM FPGA programmable connections are made using pass=transistors, transmission gates, or multiplexers that are controlled by SRAM cells. The advantage of this technology is that it allows fast in-circuit reconfiguration. The major disadvantage is the size of the chip required by the RAM technology.

Anti-Fuse Technology -- An anti-fuse resides in a high-impedance state; and can be programmed into low impedance or "fused" state. A less expensive than the RAM technology, this device is a program once device.

EPROM / EEPROM Technology -- This method is the same as used in the EPROM memories. One advantage of this technology is that it can be reprogrammed without external storage of configuration; though the EPROM transistors cannot be re-programmed in-circuit. The following table shows some of the characteristics of the above programming technologies.

Characteristics of FPGA Technology

Technology	Volatile	Re-Prog	Chip Area	R (ohm)	C (ff)
Static RAM	yes	in-circuit	large	1 -2 K	10 - 20 ff
PLICE Anti-Fuse	no	no	anti-fuse --- small ------ prog. trans.--- large	300 - 500	3 - 5 ff
ViaLink Anti-Fuse	no	no	anti-fuse --- small ------ prog. trans.--- large	50 - 60	3 - 5 ff
EPROM	no	out of circuit	small	2 - 4 k	10 -20 ff
EEPROM	no	out of circuit	2x EPROM	2 - 4 k	10 -20 ff

Table 3 Characteristics of FPGA Technology

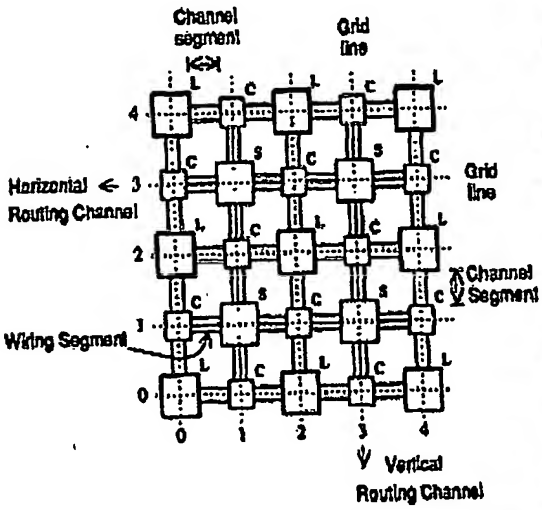
3.3 Proposed Architecture: Heterogeneous Hierarchical Routing Architecture

3.3.1 Introduction

In profiling and partitioning step recurring sequences of instructions are detected and assigned to be run on hardware. This step also takes into account the control flow structure of the code. Each bulk of pure data dependent code in between the control structures is assigned to be a zone. Then the partitioning tool runs a longest common subsequence type of algorithm to find the recurring patterns between the zones to be assigned to run on hardware. This tool also gives information about each zone such as how frequently it is used, input output pins, size, dependency between zones, etc. Placement tool threats the zones as blocks with the given information places the more related zones closer to each other. As a result the interconnection pattern is known prior to execution. That helps to exploit the locality thereby reduce the interconnection requirements.

Symmetrical FPGA (Figure 6) architecture suffers from lower density, speed and performance issues compared to conventional gate arrays. Hierarchical interconnection structures for FPGAs have been proposed to overcome these problems. The connection and switch block flexibilities may be different at each level of hierarchy. Lower levels need lesser flexibility because they have fewer connections to route. Aggarwal [AG94] says that H-FPGAs (figure 7) can implement circuits with fewer routing switches in total compared to symmetrical FPGAs. According to Li [LI99], for H-FPGAs the amount of routing resources required is greatly reduced while maintaining a good routability. It has been proved that the total number of switches in an H-FPGA is less than in a conventional FPGA under equivalent routability [LAI97]. Having fewer switches to route a net in H-FPGAs reduces the total capacitance of the network. Therefore it can implement much faster logic with much less routing resources compared to standard FPGA. H-FPGAs also offer advantages of more predictable routing with lower delays. Hence the density of H-FPGAs can be higher than conventional FPGAs. The proposed [AK02] partitioning tool analyses the application code at assembly level and extracts the

control-data flow graph. The output of the tool is a library of building blocks and their interaction. (A building block comprises of sequential but not necessarily contiguous assembly level instructions). The building blocks represent those groups or patterns of instructions that occur frequently and hence qualify for hardware implementation. Initial building blocks are formed from regions that do not contain branch instructions. These regions are referred to as zones. To maximize the use of reconfigurability amongst zones, possible parallelism and speculative execution possibilities is exploited. The Placement tool places the building blocks that are exchanging data more frequently close together. Compared to symmetrical and hierarchical fpga approach the building blocks (zones) are of variable size. Classical horizontal, vertical channel won't work in our case. Even if we inline the zones that will be a waste of space. The consistent wire bandwidth won't work because we have variable traffic with in the modules as well. Due to variable traffic among clusters and non-symmetric characteristics, different types of switches are needed at each hierarchy. Switches are variable even at the same hierarchy level. This results in heterogeneity between groups of building blocks at the same hierarchy level as opposed to classical H-FPGA approach. The aim of this chapter is to incorporate the studies mentioned above with our approach and to show a framework for the design of the heterogeneous hierarchical routing architecture.



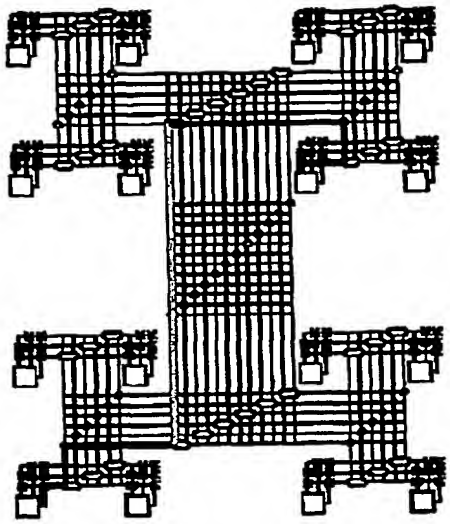


Figure 6: Symmetrical FPGA *Figure 7: Hierarchical FPGA*

3.3.2 Proposed Architecture

Clustering algorithm will give building blocks such as $B=\{B_1,B_2,\ldots B_k\}$ where $B_i \in B$. Based on the data dependency between the building blocks subsets of B which are not necessarily disjoint are formed. They are not disjoint because the dependency analysis of the building blocks might end up with the decision of inserting a building block into two subsets. If the data dependency between the building block and the candidate clusters are of the same level the clustering algorithm can't decide where to insert the building block so it provides multiple copies of the building block to the candidate clusters. As a result there possibly will be multiple copies of a building block all over the hierarchy. Clustering algorithm uses threshold-based approach to form the subsets so even though

logic blocks highly interacting each other are grouped there will be data dependency with the other clusters. When clusters are formed the data dependency, between the clusters are also kept. With this information, instead of doing an initial random placement of building blocks, now placement tool can make intelligent decision in placing the building blocks. Each building block within a cluster is placed close to each other and forms a node in a hierarchy. Having the information of data dependency between the clusters help the decision of placing the clusters efficiently to form the higher levels of hierarchy.

3.3.2.1 Overall Architecture

Clusters of building blocks form level-1 M modules. Similarly clusters of M modules form level-2 C modules. We define two types of switches as local (LS) and gateway switches(GS). Building blocks are named as $B=\{B_1,B_2,\ldots B_k\}$ where $B_i \in B$. Clusters of building blocks inside level-2 are M modules. $M=\{M_1,M_2,\ldots M_l\}$ where $Mj \in M$. Clusters of M modules inside level-3 are C modules. $C=\{C_1,C_2,\ldots C_m\}$ where $C_k \in C$. A building block B_i in module M_j of cluster C_k can establish a connection to another building block in some other cluster C_p through the local and gateway switches. On the other hand Building block B_i in module M_j of cluster C_k can establish a connection to another block within the same cluster C_k through local switches. Assuming that the clustering algorithm gave the building blocks of $B_1,B_2,\ldots B_8$ as shown in Figure-8; Figure-9 shows the overall routing architecture. Local connection is any connection within the same C module. and it uses only local type of switches(LS). If a block in module 2 of cluster 1 sends data to a block in M1 of cluster 2, data goes through the following nodes : Source Block, LS in M2, LS in C1, GS in C2, GS in Level3, GS in C2, LS in C2, LS in M1, Destination Block(figure-10).

Figure-8 Building block

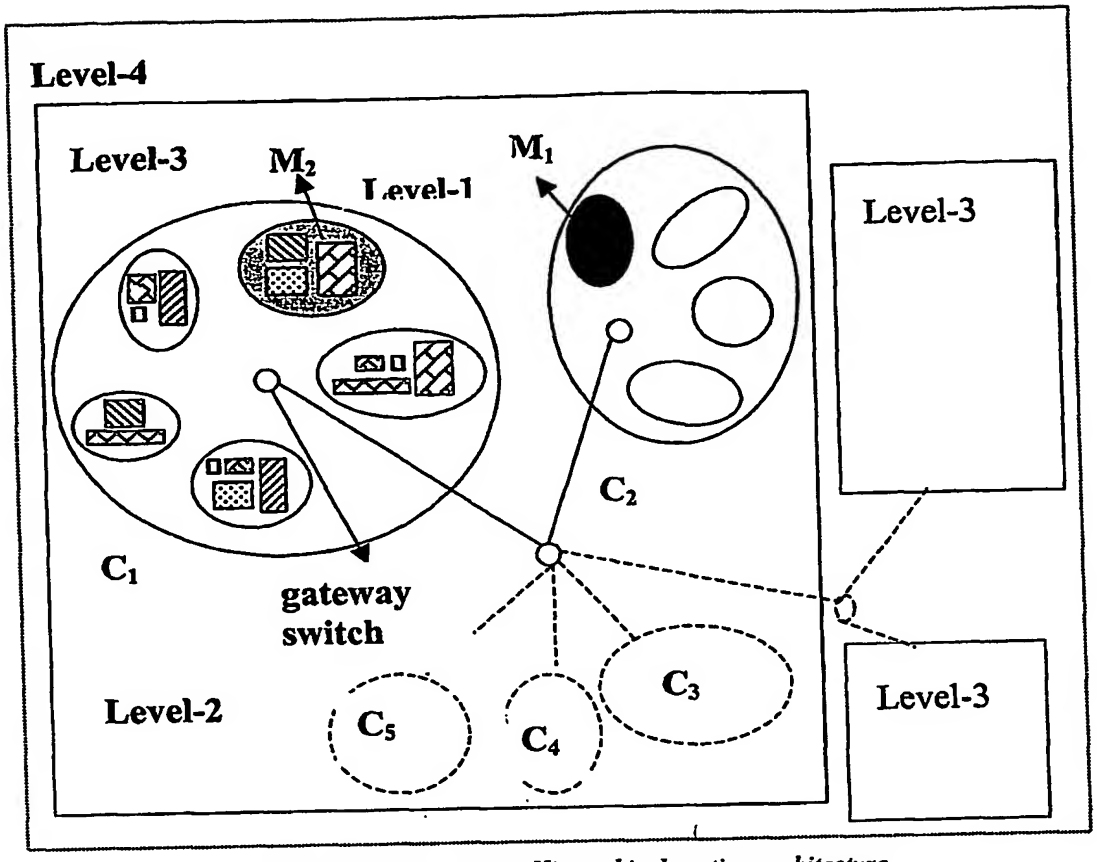


Figure 9 Heterogeneous Hierarchical routing architecture

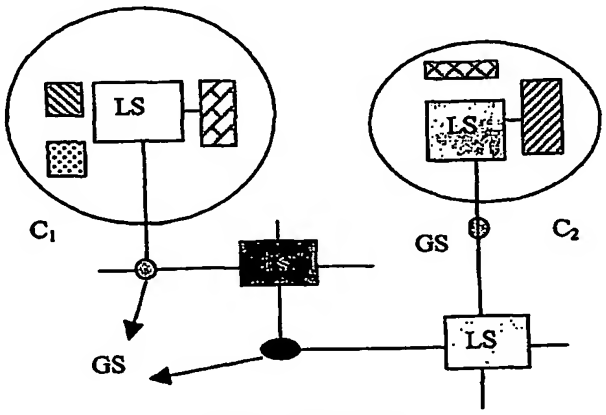


Figure-10 Switching

3.3.3 Switching Architecture

Placement knows the relationship between the zones and places the related zones closer to each other. This is the advantage over the other placement algorithm. This will help the

router in the process of decision of the switching architecture. Since we already know that the closer ones are related and more frequently changing data then the switches will be more flexible inside each cluster.
Dynamic switches are employed to handle the variable traffic load in different configurations. A switch may stay in the idle mode and process some short sequences of instruction that were not assigned to any zone. These instructions can be part of an "else" statement that is very less likely to execute so clustering tool decides not to run them on the hardware. Instead of assigning the execution of these instructions by the general-purpose processor, it is handled within the reconfigurable unit. This saves communication time and switching time. When traffic load between two modules increase after a configuration, an idle switch that are in between these modules can be activated to handle the traffic. Each switch might have copies of pins to resolve the issue of bending problems in the switches. Next section explains the procedure of how to place the building blocks that belong to the same cluster.

3.3.4 Building Block Placement

Let n be the number of building blocks in a given cluster, C_{ij} be the number of connections between building blocks B_i and B_j and D_{ij} be the amount of data traffic in terms of average number of bits transferred between the blocks B_i and B_j where $1 \le i \le n, 1 \le j \le n$. Then cost of data exchange between two library modules B_i and B_j is defined as :

$$Cost_{ij} = C_{ij} \times D_{ij}$$

Building blocks are placed based on the time to execute the circuit (T) constraint. Time is measured by the profiling tool based on the trace files specifying that the circuit should run in less than T units of time. The algorithm has three phases. Steps one and two ignores the fact the building blocks are variable in size. First step also ignores the time constraint. In the first phase placement is done on a grid style to specify if the block should be placed to north, south, east or west of another block using the dependency information. If the output of first phase matches the timing constraint then second phase is omitted and the suggested placement orientation is fed to the placement algorithm, which will allocate the actual space for the building blocks. If the timing constraint is not met, then second phase increases the wiring capacity or changes the place of building blocks till it matches the timing requirement.

3.3.4.1 First Phase :

Objective is to place the pairs of building blocks that have the most costly data exchange closest to each other. As the cost of the link decreases algorithm tolerates to have a Manhattan distance of more than 1 hop between the pairs of building blocks. This phase guarantees the best area allocation because the building blocks are placed based on their dependency leading to usage of less number of switches to establish a connection between them. Integer programming technique is used to make the decision of the orientation of the building blocks with respect to each other. Given that there are n

number of building blocks , in the worst case scenario if the blocks are placed diagonally on a grid assuming that each block is unit size of 1 then the placement is done on a nxn matrix. Let $P_i(x,y)$ denote the (x,y) coordinates of the building block B_i and no other building block has the same (x,y) coordinates. The objective function is :

$$\min\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n} f(x,y)\right)$$

$$f(x,y) = \left(\left|P_i(x) - P_j(x)\right| + \left|P_i(y) - P_j(y)\right| \times Cost_{ij}\right)$$

3.3.4.2 Second Phase:

Placement graph is the output of the first phase. That circuit's timing is measured in the second phase. If it fails then the area is compromised by either moving one or more building blocks or increasing the wiring capacity on the path from node i to node j. Obviously this will result in extra usage of area by the additional wires or switches. Algorithm runs till area and time values are below threshold. First phase gives the placement orientation information; second phase gives the switching and wiring requirements. All these information is then fed to the placement tool, which uses a modified version of simulated annealing method.

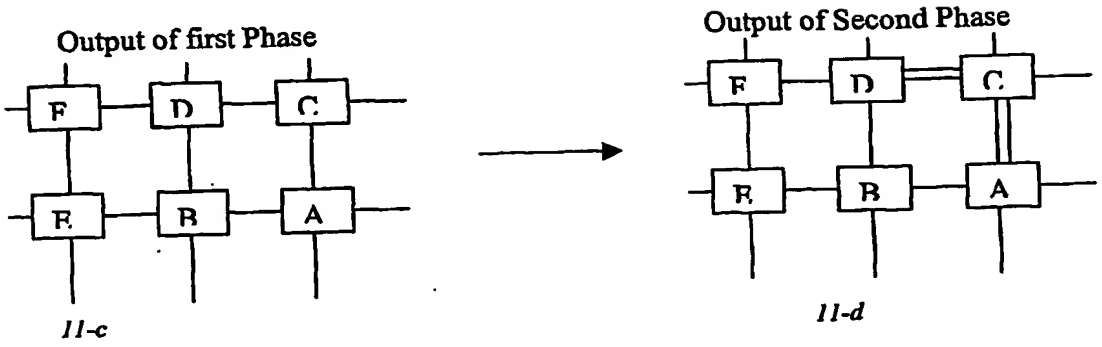
Similarly as the profiler tool gives the information about the dependency between the clusters as well, the placement at the cluster and higher levels use the same formulation explained above. Figure-11a shows the cost matrix of a given 6 blocks (A,B,C,D,E,F). Those 6 nodes are treated as points to be placed on a 6x6 matrix (figure-11b). Output of first phase is shown in figure-11c. If timing constraint is not satisfied then second phase might choose to increase the wiring capacity between the building blocks that has heavy traffic as shown in figure-11d.

Example :

Given the cost matrix and the empty grid matrix

	A	B	C	D	E	F
A	0	X	X	X	X	X
B	5	0	X	X	X	X
C	6	0	0	X	X	X
D	4	3	7	0	X	X
E	1	4	1	0	0	X
F	3	0	4	5	3	0

11-a *11-b*



11-c *11-d*

Figure 11 (a: Cost matrix b: 6x6 grid, c: Output of First Phase, d: Output of Second Phase)

3.4 Routing Algorithm

Profiling tool gives the timing analysis of each building block. The time that logic block needs to be executed is speculatively priori known. Profiling tool also gives the data dependency information between the building blocks. This information will help to reduce the complexity of finding the route between two building blocks. In addition to that we propose to allocate address table on chip, which stores the location of a C level module on chip. If a building block needs to send a data to another block it checks the address table and find out in which hierarchy level and in which C level module the destination node is located. This will help to find the place of the cluster without searching. Shortest path or best available path search algorithm is then reduced to C module level. During the reconfigurations the address table will bring the cost of update and deletion.

3.4.1 Buffering

Buffering is needed when receiver needs to process bulks of data at a time (ex: 8 rows at a time from an image) . In that case it waits till all 8 rows are filled and then starts processing the data. For a given context if we know consumer demands data in a block manner then the receiver should rearrange the incoming data format. Sender and receiver should be context aware. Buffers are only kept at the receiver side. Producer doesn't store data it simply dumps the data to the bus as soon as it is available. Receiver should be aware of the context of each request and make a decision based on the priority in order to prevent collusion. If the receiver needs to get data from more than one receiver then

those senders which are in the ok list are allowed to transmit data other requests should be denied. This is again handled by the collusion prevention mechanism. The connection service mechanism brings a control overhead cost however results in controlled router service, use of resources optimally and parallelism.

3.4.2 Possible Connection Services

3.4.2.1 Connection Oriented:

First phase is to establish connection, then transmit data and finally release connection. If producer sends data faster than the receiver can process then a flow control mechanism is needed. Sender needs to be aware of whether or not the receiver is able to keep up.

3.4.2.2 Two Way connection:

The flow of data is both directions between sender and receiver only but the receiver may not always be ready. It can be processing some other task. There can be automatic buffering and queuing done within the receiver's hardware. Second solution is consumer may send a signal to producer to stop sending but in that case buffer is needed at the producer side as well. An alternative solution is to speed up the receiver hardware by increasing the local clock frequency.

3.4.2.3 Buffering

Buffering is needed when receiver needs to process bulks of data at a time (8 rows at a time from an image) . It waits till all 8 rows are filled and then processes the data. For a given context if we know consumer demands data in a block manner then the receiver should rearrange the incoming data format. Sender and receiver should be context aware. Buffers are only kept at the receiver side. Producer doesn't store data it simply dumps the data to the bus as soon as it is available. Receiver should be aware of the context of each request and make a decision based on the priority in order to prevent collusion. If the receiver needs to get data from more than one receiver then those senders which are in the ok list are allowed to transmit data other requests should be denied. This is again handled by the collusion prevention mechanism. The connection service mechanism brings a control overhead cost however results in controlled router service, use of resources optimally and parallelism.

3.5 Conclusion

We propose that the routing tool will provide optimum interconnection pathways between different hierarchy levels with variable switching. Providing wide range of switches that has different channel widths, variable number of connections, priority switching and task multiplexing mechanisms (computation or switching) will enable cost efficient resource allocation.

3.6 A Framework for the Routability of a Dynamically Reconfigurable Processor

3.6.1 Introduction

DeHon[MIT96] showed the switching requirements using Rent's Model on Hierarchical Networks, Stroobandt[STR96] and Donath[DON79] showed the extensions of Rent's rule on the decision of Rent exponent and average interconnection length, finally Brown[BRO93] showed a Stochastic model for routability in symmetric FPGAs. According to Donath, hierarchical approach gives a better estimate of interconnection lengths. Our aim in this paper is to incorporate these studies with our approach and to show a framework for the routability with variable sized logic blocks; connection and switch block flexibilities as well as variable traffic for the dynamically reconfigurable processor. There is a need to estimate the power consumption, area and routability from a design description of the circuit. Interconnect capacitance has not scaled down as much as gate capacitance with the advances in sub micron technology. Capacitance is an important component of the power consumption with the interconnect loading. We need to incorporate the wire capacitance into power consumption. In order to do that we need to estimate the average interconnect capacitance, which requires an estimate of the average interconnect length. This measure is also necessary for the routability equation. Predicting the interconnection lengths and the total number of connections before placement is very important since this procedure reduces the requirements that help to limit the solution space for the placement algorithm and helps to choose the most suitable architecture for the design. Management of interconnection length is critical in terms of the amount of space required, time delay for signals as the capacitance increases with the wire length.

In order to find the average connection length researches have been using two ways .One is to place the building block with a given routing architecture and find the exact wiring that is necessary. However we want to predict the wiring requirement even before the placement. One solution for that is the use of Rent's rule, which defines a relationship between the building blocks forming a module and the number of external pins of that module. Rent's rule is a partitioning scheme. A net list is partitioned then each partition is partitioned again till gate level or building block level is reached.

Suppose a logic block is connected to $P= \lambda B^r$ where P is the number of external pins on a module, B is the number of blocks per module and λ is the average number of pins per block and r is called Rent's exponent which is a measure of interconnection complexity of the logic. P and r are fixed for a given logic net, therefore λ and B are the adjustable parameters. Rent's rule models the behavior of a scheme that keeps the number of interconnections between sub-designs as low as possible leading to many short interconnections and fewer long ones. Rent's rule holds equally well for partitioning of logic into hierarchical model[MIT96]. Predicting the interconnection lengths and the total number of connections before placement is very important since this procedure reduces the requirements that helps to limit the solution space for the placement algorithm and helps to choose the most suitable architecture for the design. Management of interconnection length is critical in terms of the amount of space required, time delay for

signals as the capacitance increases with the wire length. The Rent's exponent represents the interconnection complexity of the design. With the Rent exponent placement and routing predictions can be made. The exponent value ranges from 0.4 for very simple and regular interconnections up to 0.8 for complex irregular interconnections generally fast logic chips. With Rent's rule we can define the number of pins (P_i) used at the i^{th} level of the hierarchical mode. If α (total number of connections/total number of pins) is set to be a constant (obtained from the circuit) then :

Total number of connections at level i is given by $C_{Ti} = \alpha^* P_i$

Rent's rule says that a module at some hierarchy level represent the whole design so if number of external pins for that module or rent exponent for that module is found by this value can be scaled to represent the whole design. Rent's model has been applied to symmetrical fpga and hierarchical fpga approaches. These architectures use Manhattan grid style placement with fixed type of a blocks. At each level number of building blocks forming a module is fixed. It is clear that, property of one module can represent the whole design however in our case profiler tool gives out zones that form the basic building blocks and each zone is variable in terms of complexity, contents and size. Placement tool cluster the more related zones together to form a hierarchy. Each cluster is composed of different number of and different types of zones. According to Marck[MAR95], Rent's rule is not applicable if the interconnection complexities are different for blocks or hierarchical levels. In such a case average number of pins at each hierarchy level may differ, so he modified the Rent's rule by deriving local Rent exponent capable of dealing with the variations in the complexity that results with a more accurate interconnection complexity measure. We use Marck's method for deriving Rent's exponent and the stochastic model of Brown in our work. In the following sections we describe the methodology to represent the average interconnection length with Rent's Rule and the routability with stochastic model.

3.6.2 Methodology

Routability is the likelihood that a circuit can be successfully routed in a given routing architecture. Number of routing switches and their distribution over the wire segments are parameters of the stochastic model. Additional parameters such as total number of connections (C_T), length of each connection ($\overline{R}$), number of wiring tracks per channel, flexibilities of connection and switch blocks are needed to represent the circuit to be routed. However parameters C_T and $\overline{R}$ remain unknown before the actual partitioning and placement of the circuit. Rent's rule defines a relationship for partitioning logic into sub-modules.

3.6.3 Deriving Interconnection Length

In Marck's model[MAR95] nets are the interconnections between blocks through which information is transported one block to another. A net can transport information to multiple blocks. Modules are defined as a set of interconnected blocks. Each net connecting two modules are connected to pins that are the interfaces of the modules. Marc defines P_{max} as the maximum number of pins per module and M_{max} as maximum number of modules allowed. In our case, series of instructions not necessarily sequential are grouped together to form the library modules with the clustering algorithm. The algorithm uses the dependency between instructions, frequency of occurrence of patterns and control flow graph analysis information for grouping. The tool gives the library modules (programmable building blocks), dependency between blocks, frequency of occurrence of each block and number of input and output pins of each block. Cluster sizes are optimized such that the modules can execute in parallel. Since interconnection behavior between modules is known beforehand, modules that are more likely to be connected are grouped together at the same hierarchy level. This way the interconnections between library modules are reduced resulting with small number of edges crossing module boundaries. Since we have the accurate information about the modules and their interaction we don't need to define a limit on the number pins or modules. In our model complexity of building blocks and the modules at each level of the hierarchy are variable. Each module may have drastically different rent exponent and assigning a single rent exponent for the combined design will lead to wrong estimations of the properties of the layout such as the average length and total number of connections. A modification to Rent's model is needed to overcome this limitation. We do this by representing interconnection over varying complexity over the different hierarchical levels and find an exponent value for each level by:

$$P_m = CB_m^{r(B_m)}$$

$r(B_m)$ is a function of B_m.

$$r(B_m) = \log\left(\frac{P_m}{B_m}\right)$$

$$r(B_m) = \frac{\log(P_m)}{\log(B_m)}$$

The slope of the log-log diagram of $r(B_m)$ for each m gives the rent exponent.

According to Gamal[GAM81] the length of each connection is drawn from P_L which is a geometric distribution with mean length $\overline{R}$. Brown uses this information in his model. However since Gamal used fixed type of blocks in his work we can't directly assume that the length distribution in our case is geometric. Clustering tool analyzes both the static code and trace file to form clusters. That analysis calculates how many times a dependency occurs on static code. MPEG4 is highly user interactive so each mode of operation will initiate different set of building blocks to be executed. The trace files help to analyze how many times the estimated dependencies occur. These two observations provide a probability model for the expected number of occurrence of dependencies and the length We then use Brown's stochastic model to predict the routability. Procedure to find average interconnection length is as follows:

1. Total number of pins are found through the profiler tool

2. For each hierarchy level the number of connections to pin ratio is known
3. Find the average connection length for each level using Marck's model.
4. Find the average connection length for the overall design.

3.6.4 Routability Model

Routability is the ratio of expected number of successfully routed connections to the total number of connections.

C_T total number of two point connections ($C_1, C_2, \ldots, C_{CT}$)

$R_{C1}, R_{C1} \ldots R_{CCT}$ event that the connection is successfully established

$P(R_{Ci})$; probability of connection i to be successfully routed

$$Routability = \frac{\sum_{i=1}^{CT} P(R_{Ci})}{C_T}$$

Gateway switches, like local switches, have variable flexibilities over the hierarchical levels. As the hierarchy level increases the complexity of the switches will increase.

3.6.5 Probability of Successfully Routing a Connection

For W tracks in each routing channel at a specific point we define p_i as the probability of the track to be occupied after some circuit had been routed. According to Brown This event can be approximated by Poisson distribution. In Figure-5 we have shown the series of events to establish a connection between two building blocks of different C modules.

X_1: Event that the logic block pin associated with C_i can connect to at least one track at the first local switch block. (there are Fl tracks that can connect a building block but any number of those tracks may already be in use by other connections.

X_2: Event that pin associated with the first LS can connect to the local switch of the C module that the building block belongs to.

X_3: Event that pin associated wit the Ls of the C module can connect to the GS of that same C module.

$S_1, S_2, \ldots, S_n$: Events that C_i can successfully reach at least one track on the outgoing side of the first, second up to n^{th} gateway switch.

X_4: Event that GS of the C module where the destination block stays can connect to the LS of the same C module.

X_5 : Event that LS of the current C module can connect to the LS of the M module that the destination block belongs to.

X_6 : Event that LS of the M module that destination block belongs to can connect to the destination block.

LC_i is the length of connection between two blocks. Typical connection C_i is successfully routed only if all the events

$X_1, X_2, X_3, S_1, S_2 \ldots S_n, X_4, X_5, X_6$ occur.

$$P(Rc|LCi) =$$
$$P(X_1 \cap X_2 \cap X_3 \cap S_1 \cap S_2 \cap \ldots \cap S_n \cap X_4 \cap X_5 \cap X_6)$$
$$= P(X_1)P(S_1|X_1)P(S_2|S_1 \cap X_1)\ldots$$
$$P(S_n|S_{n-1} \cap \ldots \cap S_1 \cap X_1)P(X_2|S_n \cap \ldots \cap S_1 \cap X_1)$$

3.7 Conclusion and Future Work

In this chapter we present a framework to model the routability of a given routing architecture. We give an overview of the proposed tool set for the design of a dynamically reconfigurable processor and then we define the routing architecture. Based on that routing architecture we show the methodology about how to modify the Rent's model and Brown's stochastic model to predict the routability. We are currently working on the profiler tool to form the building blocks and their interaction from MPEG4 code. Next step is to list al possible events for a possible connection scenario and develop a probabilistic model for each case.

4 Testing Strategy

Circuit net list and placement file formats will be modified to compare the routing performance. A placement file is first converted into LUT format then the connectivity information between the lists of LUTs obtained is used to cluster the LUTs related to each other using the clustering tool. The output is then fed into routing tool This way we will be able to use existing benchmarks and verify the functionality of the proposed model.

5 References

(routing algorithms part)

[BOL79] H.Bolloinger A mature da system for pc layout. Proceedings of first International Printed Circuit Conference, 1979

[DK82]W.A. Dees and P.G. Karger Automated rip-up and reroute techniques, Proceseedings of Design Automation Conference, 1982

[DE86] V.K. De Aheuristic Global Router for Polycell Layout, PhD Thesis, Duke University, 1986

[LEE61]C.Y. Lee An algorithm for path connections and its applications, IRE Transactions on Electronic Computers, 1961

[Sou78] J. Soukup Fastr Maze Router, Proceedings of 15th Design Automation Conference pages 100-102, 1978

[HAD75]F. Hadlock, Finding a maxuimum cut of a planar graph in polynomial time, SIAM Journel of Computing , 4, no 3:331-225, Spet. 1975

[MT68]K. Miami and K. tabuchi, A computer program for optimal routing of printed circuit connectors, IFIPS, Proc H47:1475-1478, 1968

[HIG69] D. W. Hightower, A solution to the line routing problem on a continous plane, Proc. 6th Design Automation Workshop 19969

[BRO92] S. Brown, A detailed router fro field programmable gate, IEEE Transaction on Computer Aided Design, May 1992.

[GKG93] t.F. Gonzales, Minimization of the number of layers for single row routing with fixed street capacirty, IEE Transactions on Commmputer Aided Design, CAD 1984

[BKS92] S.Burman, Over the cell routing algorithms for industrial cell modes. Proceedings of International Conference on Computer Aided Design, 1992

[PK99] Peter Kuhn, "Complexity Analysis and Vlsi Architectures for Mpeg-4 Motion Estimation", Kluwer Academic Publishers.

[DP01] A.Dasu and S.Panchanathan, "Reconfigurable Media Processing", *IEEE,* 2001

[MP4] MPEG-4 Video Verification Model Version 18.0

[CH93] Eric Chan and Sethuraman Panchanathan, "Motion estimation Architecture for Video Compression", *IEEE Transactions on Consumer Electronics,* Vol. 39, No. 3, August 1993

[SR01] Sethuraman Ramanathan *et al.*, "HW-SW Co-Design and Verification of a Multi-standard Video and Image Codec", *IEEE,* 2001

[KJ00] S.K. Jang *et al.*, "Hardware-Software Co-implementation of a H.263 Video Codec", *IEEE,* 2000

[CH01] Chen *et al.*, "Efficient Architecture and Design of an Embedded Video Coding Engine", *IEEE Transactions on Multimedia,* Vol. 3, No. 3, September 2001.

[PK99] "Algorithms, Complexity Analysis and VLSI Architectures for MPEG 4 Motion Estimation", Peter Kuhn. Kluwer academic publishers.

[RAG00] "Complexity Analysis of MPEG-4 Video Profiles", A Master's thesis by C.N. Raghavendra. Arizona State University, 2000.

[ISO98] ISO/IEC JTC1/SC29/WG11, "MPEG-4 video verification model version 11.0," Doc. 2172, Mar. 1998.

[ARM1]http://www.arm.com/armwww.ns4/html/TRMs?OpenDocument

[HOR99] "UltraSPARC-III: designing third-generation 64-bit performance", Horel, T.; Lauterbach, G. IEEE Micro, Volume: 19 Issue: 3, May-June 1999 Page(s): 73 –85

[TRE96] "UltraSparc I: a four-issue processor supporting multimedia", Tremblay, M.; O'Connor, J.M. IEEE Micro, Volume: 16 Issue: 2, April 1996 Page(s): 42 –50

[MIPS1] http://www.mips.com/publications/processor_cores.html

[MIPS2] http://www.mips.com/publications/processor_architecture.html

[PEL96["MMX technology extension to the Intel architecture," A. Peleg and U. Weiser, IEEE Micro Mag., vol. 16, pp. 42–50, Aug. 1996.

[AK02] "Pattern Recognition Tool to Detect Reconfigurable Patterns in MPEG4 Video Processing", A. Akoglu, A. Dasu, A. Sudarsanam, M. Srinivasan, and S. Panchanathan, Workshop on Parallel and Distributed Computing in Image Processing, Video Processing, and Multimedia (PDIVM'2002), Florida, April 2002.

[DAS-2] "A Survey of Media Processing Approaches", A. Dasu and S. Panchanathan", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 12, No 8, Aug 2002. Pg (s) 633 – 645.
[DP-2] "Reconfigurable Media Processing", A. Dasu and S. Panchanathan, Parallel Computing 28 (2002). Pg (s) 1111 - 1139.
[TON96] "VIS speeds new media processing," M. Tremblay, J. M. O'Connor, V. Narayanan, and L. He, IEEE Micro Mag., vol. 16, pp. 10–20, Aug. 1996.
[KR00] "A methodology to Implement MPEG-4 Algorithms on reconfigurable Architectures", A Master's thesis by K. Ramaswamy. Arizona State University, 2000.
[DAP03] "Reconfigurable Processing", Aravind R. Dasu, Ali Akoglu and Sethuraman Panchanathan. U.S Provisional Patent Application filed on Feb/5/2003.
References for the router
[MIT96] "Reconfigurable Architectures for General-Purpose Computing." AI Technical Report 1586, MIT Artificial Intelligence Laboratory, September 1996.
[STR96] "An accurate interconnection length estimation for computer logic" *Strooband, D.; Van Marck, H.; Van Campenhout, J.;* VLSI, 1996. Proceedings., Sixth Great Lakes Symposium on , 22-23 March 1996 Page(s): 50 –55.
[DON79] "Placement and average interconnection lengths of computer logic.", W. E. Donath., *IEEE Trans. Circuits & Syst.*, CAS–26, 1979, 272–277.
[BRO93] "A stochastic model to predict the routability of field-programmable gate arrays",*Brown, S.D.; Rose, J.; Vranesic, Z.G.;* Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on , Volume: 12 Issue: 12 , Dec. 1993 Page(s): 1827 –1838.
[AG94] "Routing architectures for hierarchical field programmable gate arrays ",*Aggarwal, A.A.; Lewis, D.M.;*Computer Design: VLSI in Computers and Processors, 1994. ICCD '94. Proceedings., IEEE International Conference on , 10-12 Oct. 1994 Page(s): 475 –478.
[LI99] "Routability prediction for hierarchical FPGAs" *Wei Li; Banerji, D.K.;* VLSI, 1999. Proceedings. Ninth Great Lakes Symposium on , 4-6 March 1999.
[LAI97] "Hierarchical interconnection structures for field programmable gate arrays",*Yen-Tai Lai; Ping-Tsung Wang;* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , Volume: 5 Issue: 2 , June 1997 Page(s): 186 –196.
[GAM81] "Two-dimensional stochastic model for interconnections in master slice integrated circuits", *Gamal, A.E.;* Circuits and Systems, IEEE Transactions on , Volume: 28 Issue: 2 , Feb 1981 Page(s): 127 – 138
[MAR95] "Towards an Extension of Rent's Rule for Describing Local Variations in Interconnection Complexity ",Van Marck, H., Stroobandt, D., Van Campenhout, J., *Proceedings of the fourth International Conference for Young Computer Scientists,* pp. 136 - 141, July 1995.